修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工　学研究科　機械知能システム学　専攻　博士前期課程		
氏　　名	GAO LONGLONG	学籍番号	1732046
論 文 題 目	原始根による暗号化制御系の誤差除去		

要　　旨

情報通信技術の発展に伴い，制御系のネットワーク化が進んでいる一方で，サイバー攻撃の脅威に晒されている．盗聴防止の対策として，ElGamal 暗号を用いて制御器内部の情報を暗号化したまま制御入力を決定する暗号化制御が提案されている．暗号化する場合の制御系を本論文で暗号化制御系と呼ぶ．暗号化制御系では，元の制御系における実数値から ElGamal 暗号の扱う平文と呼ばれる非負整数値に変換する必要がある．その変換によって，暗号化する場合に決定した制御入力と元の暗号化しない場合の制御入力との間にズレが生じる．このズレを本論文で誤差と呼ぶ．誤差による制御性能への影響が予想されている．この影響をなくすには，誤差を除去する必要がある．しかし，従来手法では，原始根以外の元に基づく ElGamal 暗号が用いられ，その平文空間が非連続な非負整数集合なので，誤差を小さくできるが，完全に除去することができない．また，ElGamal 暗号は，離散対数問題の困難性を安全性の根拠とした暗号方式である．2048 ビット以上の公開鍵の ElGamal 暗号は, 2030 年まで解読されず安全であることが予想されている．そこで，本論文では，原始根に基づく ElGamal 暗号に着目し，暗号化制御系における誤差を除去する手法を提案する．原始根の場合，平文空間が連続な非負整数集合になる．また，2048 ビット以上の公開鍵の ElGamal 暗号を用いる数値シミュレーションにより，提案手法の効果を検証する．

平成30年度修士論文

原始根による暗号化制御系の誤差除去

電気通信大学

情報理工学研究科　機械知能システム学専攻

学籍番号　：　1732046

名前　：　GAO LONGLONG

指導教員　：　小木曽 公尚 准教授

副指導教員　：　新 誠一 教授

日付　：　2019年2月4日

概 要

情報通信技術の発展に伴い，制御系のネットワーク化が進んでいる一方で，サイバー攻撃の脅威に晒されている．盗聴防止の対策として，ElGamal 暗号を用いて制御器内部の情報を暗号化したまま制御入力を決定する暗号化制御が提案されている．暗号化する場合の制御系を本論文で暗号化制御系と呼ぶ．暗号化制御系では，元の制御系における実数値から ElGamal 暗号の扱う平文と呼ばれる非負整数値に変換する必要がある．その変換によって，暗号化する場合に決定した制御入力と元の暗号化しない場合の制御入力との間にズレが生じる．このズレを本論文で誤差と呼ぶ．誤差による制御性能への影響が予想されている．この影響をなくすには，誤差を除去する必要がある．しかし，従来手法では，原始根以外の元に基づく ElGamal 暗号が用いられ，その平文空間が非連続な非負整数集合なので，誤差を小さくできるが，完全に除去することができない．また，ElGamal 暗号は，離散対数問題の困難性を安全性の根拠とした暗号方式である．2048 ビット以上の公開鍵の ElGamal 暗号は，2030 年まで解読されず安全であることが予想されている．そこで，本論文では，原始根に基づく ElGamal 暗号に着目し，暗号化制御系における誤差を除去する手法を提案する．原始根の場合，平文空間が連続な非負整数集合になる．また，2048 ビット以上の公開鍵の ElGamal 暗号を用いる数値シミュレーションにより，提案手法の効果を検証する．

目 次

図 目 次

表 目 次

第1章　はじめに

本章では，ネットワーク化制御系に対するサイバー攻撃に関する背景と対策手法に関する先行研究を紹介する．その後，本研究の動機と目的を述べ，最後に，本論文の構成を示す.

1.1　背景

近年，情報通信技術や新技術の発展により，従来閉じたネットワークで運用されていた工場，水道・ガス・電気など，現代社会を支える重要インフラにおける制御系のネットワークがインターネットなど外部ネットワークに接続する機会が増えている．特にものづくり産業の IoT などが進むにつれて，その傾向が顕著になっている．これらにより，制御系を対象としたサイバー攻撃の脅威も増加している[1)].

過去の事例として，2005 年に，ダイムラー・クライスラー（現ダイムラー）の米国にある 13 の自動車工場がウイルス感染により操業停止となる事故が発生した．外部から持ち込まれ、制御システムに接続されたノート PC 経由の可能性が指摘されている．各工場のシステムはオフラインになり，組み立てラインで働く 50000 人の労働者は作業中断を余儀なくされ，自動車生産が 50 分間停止する状態となった．部品サプライヤへの感染も疑われ部品供給の懸念も生じ，およそ 1400 万ドルの損害をもたらした[2)]．2014 年に，ウクライナの電力システムがサイバー攻撃を受け，甚大な被害を被っている[3)]．その他，2017 年にホンダの国内外のコンピューターネットワークがサイバー攻撃を受け、乗用車アコードなどを生産している狭山工場（埼玉県）の操業を 19 日に一時停止した[4)]．これを受け，制御理論に基づくアプローチとしての制御システムへの攻撃検知や，攻撃に対する性能保証などに関する研究が進んでいる一方で[5, 6)]，情報工学分野における暗号理論を応用したものなども続々提案されている．

1.2　先行研究

ネットワーク化制御系は，従来の情報システムと同じように，常に盗聴の危険に晒されている．その対策として，暗号技術による秘匿手法が考案されている．例えは，制御系における通信路中の信号を暗号化し，信号盗聴を防ぐ手法が提案されている[7]．また，制御システムの周期性・リアルタイム性を満たすために，制御用コントローラ向けの高速化手法が考えられている[8]．しかし，これらの研究では，通信路においてのみ暗号化が行われているので，制御器への不正侵入が行われた場合，制御器内部の暗号化されていない情報が奪取される可能性がある．

これを受け，2015 年に制御器内部のパラメータや信号などの情報を，乗法準同型暗号 ElGamal と RSA を用いて暗号化したまま演算し，制御入力を決定する手法が初めて提案された．この手法は暗号化制御と呼ばれる[9]．その後，オブザーバを併合した制御システムに対する暗号化制御が考えられている[10]．暗号化制御のリアルタイム性については，多倍長整数ライブラリを用い，128 ビット以下の公開鍵を用いた場合，リアルタイム性を満たすことが実験装置で検証された[11]．そして，暗号化制御の場合，実数から整数への変換誤差が生じ，制御系への影響が懸念されている．鍵長を長くすることによって，その影響が一定の範囲に収まることが確認された[12]．

一方，乗法準同型暗号の他に，加法準同型性を持つ Paillier 暗号は，線形制御系[13]や最適化問題[14, 15, 16]などへの適用に関する研究も活発である．それに，平文の加算と乗算の両方が可能な完全準同型暗号を用いた暗号化制御も提案された[15]．

1.3　動機と目的

ElGamal 暗号を用いて制御器の内部情報を暗号化する場合の制御系は，本論文で暗号化制御系と呼ぶ．暗号化制御系では，元の制御系における実数値から ElGamal 暗号の扱う平文と呼ばれる非負整数値に変換する必要がある．その変換によって，暗号化する場合に決定した制御入力と元の暗号化しない場合の制御入力との間にズレが生じる．このズレを本論文で誤差と呼ぶ．誤差による制御性能への影響が予想されている．この影響をなくすには，誤差を除去する必要がある．しかし，従来手法では，原始根以外の生成元に基づく ElGamal 暗号が用いられ，その平文空間が非連続な非負整数集合なので，誤差を小さくできるが，完全に除去することができない．また，ElGamal 暗号は，離散対数問題の困難性を安全性の根拠とした暗号方式である[17]．2048 ビット以上の公開鍵の ElGamal 暗号は，2030 年まで解読されず安全であることが予想されている[18]．

そこで，本論文の目的は，暗号化制御系における誤差を除去する手法を提案することである．これにより，攻撃者がネットワークに繋がる制御器に侵入した場合，情報を保護することができる同時に，制御性能に影響を及ばさないことがで

きる．

研究目的を達成するため，本論文では，生成元が原始根である ElGamal 暗号に着目する．原始根の場合，平文空間が連続な非負整数集合になる．これに基づき誤差を除去する手法を提案する．また 2048 ビット以上の公開鍵の ElGamal 暗号を用いる数値シミュレーションにより，提案手法の効果を検証する．

1.4　本論文の構成

本論文の構成は，以下のとおりである．まず，第 2 章では，本論文で扱う原始根に基づく ElGamal 暗号の原理や理解に必要な数学の基礎知識を解説する．第 3 章では，比例制御器に対して，誤差を除去する提案手法を説明する．第 4 章では，提案手法の効果について，数値シミュレーションによる検証を行う．また，提案手法の計算時間を調べる．第 5 章では，本研究のまとめおよび今後の課題について述べる．

第2章　数学準備

本章では，原始根に基づく ElGamal 暗号の原理や理解に必要な数学の基礎知識を説明する．まず暗号理論に関する各種用語を説明し，次に本論文で取り扱う原始根に基づく ElGamal 暗号の構成，準同型性，安全性及び特徴について解説する．本論文では以下の表記を用いる．

$\mathbb{N}$: 自然数全体（0 を含まない）.
$\mathbb{Z}$: 整数全体.
$\mathbb{Z}^+$: 非負整数全体（0 を含む）.
$\mathbb{Z}_n$: 自然数 n を法とする $\mathbb{Z}$ の剰余系 $\mathbb{Z}_n = \{0, 1, \cdots, n-1\}$.
$\mathbb{Z}_n^*$: 自然数 n を法とする $\mathbb{Z}$ の既約剰余系.
$\mathbb{R}$: 実数全体.
$\mathbb{R}^+$: 非負実数全体（0 を含む）.
$\mathcal{M}$: 平文空間（平文の集合）.
$\mathcal{C}$: 暗号文空間（暗号文の集合）.
$\gcd(a, b)$: $a, b \in \mathbb{N}$ として，a と b の最大公約数.
$\mathrm{lcm}(a, b)$: $a, b \in \mathbb{N}$ として，a と b の最小公倍数.
$\varphi(x)$: $x \in \mathbb{N}$ として，x のオイラー関数.

2.1　諸定義

2.1.1　原始根

本節では，原始根の定義と判定法について解説する．

定義

原始根は，巡回群に基づくものである．まず，準備として群についての基礎知識を示す.

群 (group)$\mathbb{G}$ とは，以下の規則を満たす 2 項演算 $\circ$ をもつ集合のことである.

- [結合法則] すべての元 $a, b, c \in \mathbb{G}$ に対し，$(a \circ b) \circ c = a \circ (b \circ c)$ が成立.
- [単位元の存在] すべての元 $a \in \mathbb{G}$ に対し，$a \circ e = e \circ a = a$ が成立するような元 $e \in \mathbb{G}$ が存在する．このような元 e を**単位元**とよぶ.
- [逆元の存在] すべての元 $a \in \mathbb{G}$ に対し，$b \in \mathbb{G}$ が存在して，$a \circ b = e$．このような元 b を a の**逆元**とよぶ.

群の元の数を **群の位数** とよぶ．群である $\mathbb{G}$ のすべての元がその 1 つの元 g によって生成されるとき，$\mathbb{G}$ を**巡回群**という．このような元 g を $\mathbb{G}$ の**生成元**という．$\mathbb{G} := \langle g \rangle$ と表す．p が素数ならば $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$ は，位数が $p-1$ の乗法に関する巡回群となる.

$\mathbb{Z}_p^*$ に対して，**元の位数**を，$a^m = 1 \mod p$ となる最小の自然数 m と定義する．それに，a の位数は，$p-1$ を割り切る．すなわち，$\mathbb{Z}_p^*$ に属する元は，$p-1$ を割り切る位数をもつことである.

位数 $p-1$ をもつ元 a を，$\mathbb{Z}_p^*$ の**原始根（元）**とよぶ．$\mathbb{Z}_p^*$ に属する原始根以外の元で生成された巡回群は $\mathbb{Z}_p^*$ の**部分巡回群**になる.

判定法

任意の素数 p に対して，$\mathbb{Z}_p^*$ の原始根は $\varphi(\varphi(p))$ 個存在し，その役割は約 0.6 であることが知られている．$\mathbb{Z}_p^*$ の原始根を求める時，次の判定法を利用できる [19].

素数 p に対して，$p-1$ の素因数分解に現れる素数を $p_1, p_2, \cdots, p_k$ とするとき，整数 a に対して，

$$a^{(p-1)/p_i} \neq 1 \mod p$$

がすべての $i = 1, 2, \cdots, k$ について成り立つならば，a は $\mathbb{Z}_p^*$ の原始根である.

2.1.2　離散対数問題と DDH 仮定

本節では，離散対数問題と DDH 仮定について述べる [20].

離散対数問題

p を素数，g を $\mathbb{Z}_p^*$ の原始根とする．そのとき，$\mathbb{Z}_p^*$ の任意元 y に対して，次を満たす x が存在する.

$$y = g^x \mod p$$

このとき，x を解く問題を，**離散対数問題**（DLP：Discrete Logarithm Problem）という．p が十分に大きい場合には，離散対数問題を効率的に解けない.

DDH 仮定

DDH 問題（DDH：Decisional Diffe-Hellman）とは，p を素数，$G = \langle g \rangle$ を $\mathbb{Z}_p^*$ の部分巡回群，素数 q を G の位数として，$a, b, c \in \mathbb{Z}_q$ をランダムに選択した時，(g, g^a, g^b, g^{ab}) と (g, g^a, g^b, g^c) を識別する問題である．

q が十分に大きい場合には，DDH 問題を効率的に解けない．DDH 問題を効率的に解くアルゴリズムは存在しないという仮定を，**DDH 仮定**という．

離散対数問題が解ければ，DDH 問題が解ける．しかし，その逆が言えるかどうかは未解決である．

2.1.3　平方剰余

p を奇素数（2 以外の素数）とし，a を $\gcd(a, p)$=1 を満たす整数とする．このとき，$x^2 = a \mod p$ が解を持つとき，a は法 p の**平方剰余**になる．解を持たないとき，a は法 p の平方非剰余になる．これを平方剰余記号（ルジャンドル記号）という記号で定義する．

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & (a\text{：平方剰余}) \\ -1 & (a\text{：平方非剰余}) \end{cases}$$

平方剰余記号について，次に示す性質が成り立つことが知られている．

- $\gcd(a, p) = 1$，$\gcd(b, p) = 1$ のとき，$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$
- $a = b \mod p$ であれば，$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$

2.1.4　準同型暗号

本節では，準同型暗号について説明する．まず，準備として公開鍵暗号についての諸定義を示す．

暗号とは，送信者がメッセージを別の形にすることにより，第三者には秘匿し，意図した受信者だけが判読できるようにする秘密通信の手段のこと，となる．元のメッセージそのものを**平文**，第三者に秘匿する形にしたメッセージを**暗号文**という．平文の集合を平文空間，暗号文の集合を暗号文空間という．平文を暗号文に変換する過程を**暗号化**，その逆の過程を**復号化**という．

暗号化の際に用いられる鍵を**暗号化鍵**といい，復号化の際に用いられる鍵を**復号化鍵**という．暗号化鍵と復号化鍵とが一致している暗号を**対称暗号**，または**共通暗号**といいい．暗号化鍵と復号化鍵とが異なる暗号を**非対称暗号**といい．対称暗号ではその共通な鍵を秘密にする必要があるが，非対称暗号では暗号化鍵は公開しても構わない．そこで，対称暗号，非対称暗号，それぞれ，**秘密鍵暗号**，**公開**

鍵暗号とも呼ぶ．そして，公開鍵暗号では，暗号化鍵，復号化鍵，それぞれ，**公開鍵**，**秘密鍵**とも呼ばれる．

準同型暗号は以下のように定義される [21]．

定義 2.1.1. 公開鍵暗号方式 (Gen，Enc，Dec) が準同型性をもつとは，以下を満たすことである．

- 平文空間 $\mathcal{M}$ と暗号文空間 $\mathcal{C}$ が群となっている．群の演算子はそれぞれ $\circ$ と $*$ とする．
- 任意の $m \in \mathcal{M}$ に対し，$\mathsf{Enc}(m) \in \mathcal{C}$ である．
- 任意の $m_1, m_2 \in \mathcal{M}$ と $c_1, c_2 \in \mathcal{C}$，ただし $c_1 = \mathsf{Enc}(m_1)$ かつ $c_2 = \mathsf{Enc}(m_2)$，に対し

  $$c_1 * c_2 = \mathsf{Enc}(m_1 \circ m_2)$$

  が成立する．

準同型性をもつ公開鍵暗号を**準同型暗号**という．

この準同型性は，平文空間における群演算 $\circ$ は，乗算の場合は，**乗法準同型性**，加算の場合は，**加法準同型性**，そして乗算と加算の両方の場合は，**完全準同型性**とも呼ばれる．

2.2　原始根に基づく ElGamal 暗号

原始根に基づく ElGamal（エルガマル）暗号は，1984 年にエルガマルによって提案され，巡回群 $\mathbb{Z}_p^*$ に基づく乗法準同型暗号方式である [17]．

2.2.1　構成

原始根に基づく ElGamal 暗号は，鍵生成 Gen，暗号化 Enc，復号化 Dec の組 (Gen，Enc，Dec) で構成される．

- Gen：公開鍵 $pk := (p, g, h)$ と秘密鍵 $sk := s$ は以下の手順で生成される．
  1. k ビットの素数 p を生成する．
  2. $\mathbb{Z}_p^*$ の原始根 g を選ぶ．$\mathcal{M} = \mathbb{Z}_p$.
  3. $s \in \mathbb{Z}_p$ をランダムに選ぶ．
  4. $h = g^s \mod p$ を計算する．

- Enc：平文 $m \in \mathcal{M}$ を暗号化し，暗号文 $c := (c_1, c_2)$ を求める．
  1. $r \in \mathbb{Z}_p$ をランダムに選ぶ．
  2. $c_1 = g^r \mod p$, $c_2 = mh^r \mod p$ を計算する．
- Dec：暗号文 c を復号化し，復号結果 m' を求める．
  1. $m' = c_2 c_1^{p-1-s} \mod p$ を計算する．

原始根に基づく ElGamal 暗号の正当性は，次のように証明できる．正当性とは，暗号文を復号化すると，元の平文に戻ることである．

$$\begin{aligned} m' &= c_2 c_1^{-s} \mod p \\ &= c_2 c_1^{p-1-s} \mod p \\ &= m g^{rs} g^{r(p-1-s)} \mod p \\ &= m (g^{(p-1)})^r \mod p \\ &= m (1)^r \mod p \\ &= m \mod p \\ &= m \end{aligned}$$

2.2.2　乗法準同型性

原始根に基づく ElGamal 暗号は乗法準同型性をもつ．m_1，m_2 に対する暗号文は次の通りにする．

$$\begin{aligned} \mathsf{Enc}(m_1) &= (c_{11}, c_{12}) = (g^{r_1},\ m_1 h^{r_1}) \\ \mathsf{Enc}(m_2) &= (c_{21}, c_{22}) = (g^{r_2},\ m_2 h^{r_2}) \end{aligned}$$

暗号文空間における群演算 $*$ を成分ごとの積として定義すれば，

$$\begin{aligned} \mathsf{Enc}(m_1) * \mathsf{Enc}(m_2) &:= (c_{11}c_{21},\ c_{12}c_{22}) \\ &= (g^{r_1} g^{r_2},\ m_1 h^{r_1} m_2 h^{r_2}) \\ &= (g^{r_1 + r_2},\ m_1 m_2 h^{r_1 + r_2}) \\ &= \mathsf{Enc}(m_1 m_2) \end{aligned}$$

となり，乗法準同型であることがわかる．これにより，乗算の積は，次のように，乗数と被乗数の暗号文から求められる．

$$m_1 m_2 = \mathsf{Dec}(\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2)) \quad m_1,\ m_2,\ m_1 m_2 \in \mathcal{M}$$

2.2.3　安全性

原始根に基づく ElGamal 暗号の安全性は，離散対数問題の困難性を根拠とする．離散対数問題が解ければ，ElGamal 暗号は破られる．そのため，素数である公開鍵 p は，離散対数問題が解けないくらい大きくなければならない．現在，p が 2048 ビット以上の時，離散対数を求めることが困難であり安全とされている [18]．

しかし，原始根に基づく ElGamal 暗号は，DDH 仮定が破れ，部分情報が漏えいする．以下に暗号文から，平文の部分情報が漏えいすることを証明する．攻撃者は h に対して平方剰余記号を適用して，1 になるか -1 になるかを調べる [20]．

- $(\frac{h}{p}) = 1$ の場合

  c_2 に対して平文剰余記号を適用すると，次のように展開できる．

  $$(\frac{c_2}{p}) = (\frac{mh^r}{p}) = (\frac{m}{p})(\frac{y}{p})^r = (\frac{m}{p})$$

  つまり，暗号文 c から，平文 m が法 p の平方剰余か否かの情報が漏れている．

- $(\frac{h}{p}) = -1$ の場合

  c_1 に対して平方剰余記号を適用すると，次のように展開できる．

  $$(\frac{c_1}{p}) = (\frac{g^r}{p}) = (\frac{g}{p})^r$$

  ここで，c_1，p，g は既知なので，$\frac{c_1}{p}$ と $\frac{g}{p}$ を計算できる．そこで，r の偶奇を特定する．

  c_2 に対して平方剰余記号を適用すると，次のように展開できる．

  $$(\frac{c_2}{p}) = (\frac{mh^r}{p}) = (\frac{m}{p})(\frac{y}{p})^r = (\frac{m}{p})(-1)^r$$
  $$(\frac{m}{p}) = (\frac{c_2}{p})(-1)^r$$

  r はすでに特定済みなので，暗号文 c から平文 m が法 p の平方剰余か否かの部分情報が漏れている．

以上より，原始根に基づく ElGamal 暗号では平文 m が法 p の平方剰余か否かという部分情報が漏えいするが，平文 m そのものが漏えいされていないことがわかる．

2.2.4　特徴

エルガマルが最初に提案した ElGamal 暗号は $\mathbb{Z}_p^*$ の原始根に基づくものである．原始根の場合，平文空間 $\mathcal{M} = \mathbb{Z}_p = \{0, 1, ..., p-1\}$ は 0 から $p-1$ までの連続な非負整数の集合になる．しかし，平文 m が法 p の平方剰余か否かという部分情報が漏えいする弱点がある．

それに対して，先行研究での ElGamal 暗号は，$\mathbb{Z}_p^*$ の原始根ではない元に基づくものである．この場合，平文の部分情報が漏えいしないが，平文空間 $\mathcal{M}$ は，$\mathbb{Z}_p^*$ の部分巡回群なので，非連続な非負整数の集合になる．

本論文では，原始根に基づく ElGamal 暗号の連続な平文空間に着目し，誤差を除去する手法が考えられる．

第3章　誤差除去

本章では，本論文の提案手法について解説する．まず問題設定を示し，次に原始根に基づく ElGamal 暗号を用い，誤差を除去する提案手法を説明する．最後に変換ゲインの満たす条件を示す．

3.1　問題設定

本論文では，Fig. 3.1 に示めように，ネットワークに繋がる事前に設計された離散時間制御器は，以下の制御則 f により記述されるものとする．

$$u(t) = f(K, y(t)) = Ky(t), \quad t \in \mathbb{Z}$$

ここで，t は，ステップである．$K \in \mathbb{R}$ は，定常比例ゲインで，小数部の桁数が $a \in \mathbb{Z}^+$ の有限小数とする．$y(t) \in \mathbb{R}$ は，制御対象の出力で，小数部の桁数が $b \in \mathbb{Z}^+$ の有限小数とする．$u(t) \in \mathbb{R}$ は，制御入力である．

攻撃者が上記の制御器へ侵入した場合，K，$y(t)$ と $u(t)$ が盗聴される可能性がある．盗聴を防ぐため，K と $y(t)$ を暗号化したまま，$u(t)$ を求める方法が考えられる．

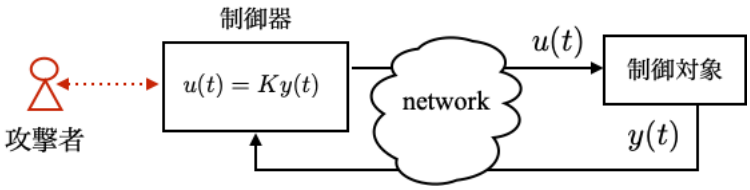


Fig. 3.1: 制御器の内部情報の盗聴

仮定の合理性

制御系で扱う実数値の信号は，理論的には，小数が無限に続く値であるが，コンピュータの内部では，有限個の桁数でその値を表すしかない．そのため，K と $y(t)$ を有限小数とするのが合理的であると思われる．

3.2　誤差除去

本節では，原始根に基づく ElGamal 暗号を用いて，K と $y(t)$ を暗号化したまま $u(t)$ を求める提案手法を説明する．暗号化の場合に求められた制御入力を $\bar{u}(t)$ で表す．準備として，まず符号付きの乗算の扱いと有限小数と整数との間の変換について解説する．

符号付きの乗算の扱い

符号付き数の乗算を取り扱う方法には，全て正の数に変換して乗算を行い，結果に符号をつけるという方法がある．式で表せば，次のようになる．

$$u(t) = Ky(t) = (-1)^{s_K \oplus s_y}|K||y(t)|$$

ここで，$s_K = \begin{cases} 0 & (K \geq 0) \\ 1 & (K < 0) \end{cases}$，$s_y = \begin{cases} 0 & (y(t) \geq 0) \\ 1 & (y(t) < 0) \end{cases}$，$\oplus$ は排他的論理和の記号，$|\cdot|$ は絶対値の記号である．

有限小数と整数との間の変換

有限小数は適当な 10 の倍数をかけると，整数になれる．そして，その倍数を除算すると，元の有限小数に戻る．よって，有限小数 x の小数部の桁数を $n \in \mathbb{Z}^+$，10 の倍数を変換ゲイン $\gamma = 1 \times 10^i (i \in \mathbb{Z}^+)$ としたとき，次式が成り立つ．

$$x = (\gamma x)/\gamma,\ \gamma x \in \mathbb{Z}$$

ただし，$\gamma \geq 1 \times 10^n$.

上記のもと，$\bar{u}(t)$ は次式で求められる．

$$\bar{u}(t) = (-1)^{s_K \oplus s_y}\mathsf{Dec}(\mathsf{Enc}(\gamma_K|K|) * \mathsf{Enc}(\gamma_y|y(t)|))/(\gamma_K\gamma_y) \tag{3.1}$$

ただし，

$$\gamma_K \geq 1 \times 10^a$$
$$\gamma_y \geq 1 \times 10^b$$

$u(t) = \bar{u}(t)$ は，次のように証明する．

$$\begin{aligned} u(t) &= Ky(t) \\ &= (-1)^{s_K \oplus s_y} |K||y(t)| \\ &= (-1)^{s_K \oplus s_y} \gamma_K |K| \gamma_y |y(t)| / (\gamma_K \gamma_y) \\ &= (-1)^{s_K \oplus s_y} \mathsf{Dec}(\mathsf{Enc}(\gamma_K |K|) * \mathsf{Enc}(\gamma_y |y(t)|)) / (\gamma_K \gamma_y) \\ &= \bar{u}(t) \end{aligned}$$

よって，式 (3.1) により，K と $y(t)$ を暗号化したまま $u(t)$ を求めることがわかる．即ち誤差 $u(t) - \bar{u}(t)$ を除去することができる．

次に，盗聴の場合，上記の提案手法による制御器内部の情報を保護できることを示す．式 (3.1) を実装するとき，次のような処理段階を分けられる．

- 鍵生成
- 有限小数から非負整数への変換
- 暗号化
- 暗号文同士の演算
- 復号化
- 非負整数から有限小数への逆変換

鍵生成は，オフラインで事前に処理できるが，それ以外の処理は，Fig.3.2 に示す順番のように，オンラインで行わなければならない

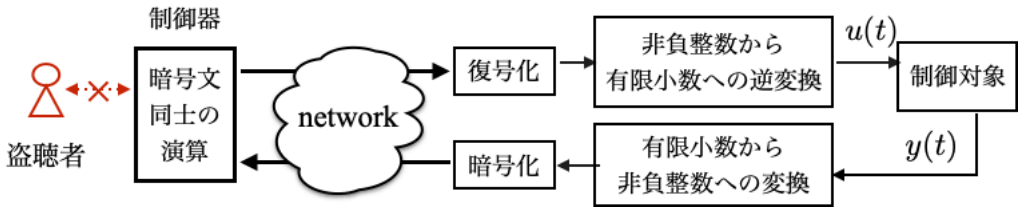


Fig. 3.2: 制御器の内部情報の盗聴防止

制御器においては，暗号文同士の演算 $\mathsf{Enc}(\gamma_K |K|) * \mathsf{Enc}(\gamma_y |y(t)|)$ が行われ，そして，γ_K，γ_y，s_K と s_y の信号が流れている．K と $y(t)$ の符号以外の情報は暗号化され，盗聴の防ぎができることがわかる．

3.3　変換ゲインの条件

これまで，式 (3.1) が成り立つための $\gamma_K \geq 1 \times 10^a$ と $\gamma_y \geq 1 \times 10^b$ という下限条件を示したが，変換ゲインについては，上限条件も存在する．式 (3.1) は，ElGamal の次に示すような乗法準同型性に基づくので，

$$m_1 m_2 = \mathsf{Dec}(\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2)) \quad m_1,\, m_2,\, m_1 m_2 \in \mathcal{M}$$

次の三つの条件を満たさなければならない．

$$\gamma_K |K| \in \mathcal{M}$$
$$\gamma_y |y(t)| \in \mathcal{M}$$
$$\gamma_K |K| \gamma_y |y(t)| \in \mathcal{M}$$

平文空間 $\mathcal{M} = \{0,\, 1,\, ...,\, p-1\}$ の最大値 $p-1$ は，$2^k - 1$ なので，上記三つの条件を満たす充分条件として，次のように挙げられる．

$$\gamma_K |K| < 2^{\frac{k}{2}} - 1,\ \ \gamma_y |y(t)| < 2^{\frac{k}{2}} - 1 \tag{3.2}$$

k は本論文で 2048 ビット以上の大きなな数値なので，式 3.2 は，一般的に満足される．

第4章　数値例

本章では，第 3 章で提案した手法を検証するため，数値シミュレーションを行う．まずシミュレーションの対象となる制御系の設定を示す．次に提案手法のシミュレーション結果を述べる．最後に提案手法を実装するときの計算時間について調べる．

4.1　制御系の設定

数値例では，フィードバック制御系のレギュレータ (定値制御) 問題を考える．制御対象は，以下に示す連続時間線形システム

$$\begin{cases} \dot{x}_p &= \begin{bmatrix} 0 & 2 \\ -2 & -3 \end{bmatrix} x_p + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_p \end{cases}$$

とする．ここで，x_p は，制御対象の状態変数を表す．初期状態は，$x_p(0) = [1\ 0]^\top$ とし，この初期状態に対して，出力のレギュレーション問題 $y(t) \to 0$ を考える．制御器は，以下の比例制御を考える．

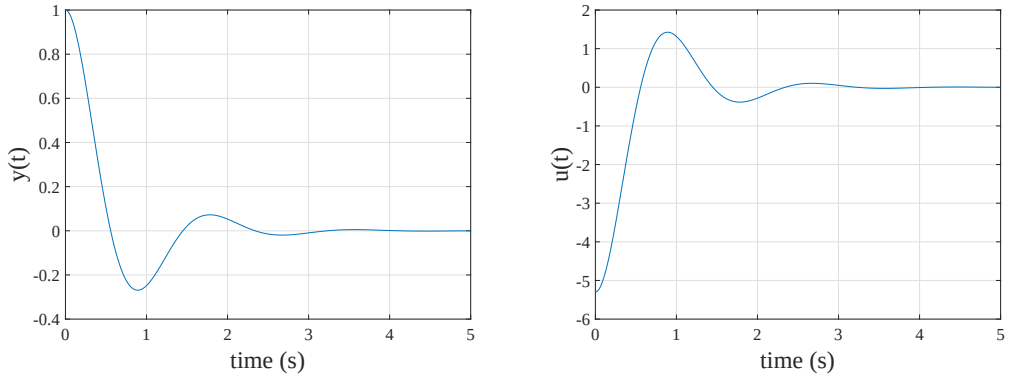
$$u(t) = f(K, y(t)) = Ky(t)$$

ここで，比例ゲイン $K = -5.3$ とする．

上記のフィードバック制御系は，零次ホールド法を用い，離散化間隔 10^{-2}[s] で離散化すると，次の離散時間線形制御系が得られる．

$$\begin{cases} x_p(t+1) &= \begin{bmatrix} 0.999802 & 0.019702 \\ -0.019702 & 0.970250 \end{bmatrix} x_p(t) + \begin{bmatrix} 0.000099 \\ 0.009851 \end{bmatrix} u(t) \\ \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_p(t) \end{cases} \tag{4.1}$$

$$u(t) = -5.3y(t) \tag{4.2}$$



(a) 出力　　(b) 制御入力

Fig. 4.1: 出力と制御入力の応答

出力 $y(t)$ と制御入力 $u(t)$ の応答を，Fig.4.1 に示す．出力 $y(t)$ を小数部の桁数が 6 である有限小数としたとき，暗号化しない場合の制御入力 $u(t)$ を，Table 4.1 に示す．

4.2　シミュレーション結果

本節では，第 3 章で提案した手法の数値シミュレーションを行う．Table 4.1 に示す $y(t)$ と $K = -5.3$ を，2048 ビットの公開鍵の ElGamal 暗号により暗号化したまま，0s から 5s までの $\bar{u}(t)$ を求める．

まず，$t = 0.1$s の時の $K = -5.3$ と $y = 0.934467$ から，$\bar{u}$ を求めるのを説明する．ソースコードは付録 A を参考されたい．

Table 4.1: 暗号化しない場合の制御入力 $u(t)$

t(s)	$y(t)$	$u(t)$	t(s)	$y(t)$	$u(t)$
0	1	-5.3	2.6	-0.018654	0.0988662
0.1	0.934467	-4.9526751	2.7	-0.019377	0.1026981
0.2	0.768934	-4.0753502	2.8	-0.017496	0.0927288
0.3	0.549598	-2.9128694	2.9	-0.013909	0.0737177
0.4	0.317587	-1.6832111	3.0	-0.009500	0.05035
0.5	0.105116	-0.5571148	3.1	-0.005030	0.026659
0.6	-0.066262	0.3511886	3.2	-0.001072	0.0056816
0.7	-0.185627	0.9838231	3.3	0.002009	-0.0106477
0.8	-0.251344	1.3321232	3.4	0.004053	-0.0214809
0.9	-0.268894	1.4251382	3.5	0.005070	-0.026871
1.0	-0.248377	1.3163981	3.6	0.005193	-0.0275229
1.1	-0.202072	1.0709816	3.7	0.004636	-0.0245708
1.2	-0.142343	0.7544179	3.8	0.003643	-0.0193079
1.3	-0.080077	0.4244081	3.9	0.002447	-0.0129691
1.4	-0.023699	0.1256047	4.0	0.001251	-0.0066303
1.5	0.021251	-0.1126303	4.1	0.000204	-0.0010812
1.6	0.052075	-0.2759975	4.2	-0.000600	0.00318
1.7	0.068529	-0.3632037	4.3	-0.001125	0.0059625
1.8	0.072223	-0.3827819	4.4	-0.001375	0.0072875
1.9	0.065953	-0.3495509	4.5	-0.001390	0.007367
2.0	0.053045	-0.2811385	4.6	-0.001227	0.0065031
2.1	0.036805	-0.1950665	4.7	-0.000953	0.0050509
2.2	0.020113	-0.1065989	4.8	-0.000629	0.0033337
2.3	0.005167	-0.0273851	4.9	-0.000309	0.0016377
2.4	-0.006608	0.0350224	5.0	-0.000033	0.0001749
2.5	-0.014554	0.0771362			

鍵生成

関数 KeyGen() により，2048 ビットの p に基づく公開鍵 $pk = (p, g, h)$ と秘密鍵 $sk = s$ を生成する（十六進数）.

$$
\begin{aligned}
p =& a81a02f3da153e9167455930977d7bd830251137e5c5c184567d1219dac\\
& fe9c057da3d8aa4bbc7e1a67a15e4818ee1085b94451e8cb3de9cd50cd49\\
& f68cda2aaa378c72e673de14c9fb054eba487e567bf09af765f6379e498\\
& 3dd01416e338994ba5f21225676a83f2f2c7551d49606e6fef9674a2c2\\
& 0c951cc9e3fb7fb04dd132ef15dfc32ebdd1e5b6a614ce64cf2fdab14492\\
& 5c8010a7ac7d30f5d02df4f1b4db902e8240b97c4fcbe0498c386eab1abc\\
& 63e39dc0e33668ad76b7aee751e79306a3bf56a68c1b3eadeab2c6e8472f\\
& 5384f59dcd9c70be74b728f8040f5862e188c8c4783456eb386e98a7058\\
& 6e4ad312ed3a95b35c86aa649d9ab42d07e77\\
g =& 2\\
h =& 3cd5558d99def8306dc3249667f8102b06d2f3a62955145d7441fe92e21\\
& 6e093fc3edb1270f9bc6c4058a3c3b56d4fb70d9e8edf877d5a8c393adc65\\
& 350f797c21b462f32221b5e77643f14ffa43d90b002fdfe38df7fe7303c4\\
& f5eab67347c4bb6bfa80b5b0617c6c078d570df222285fb24796532d31b94\\
& 07e9978838ec82b4711ccb07f2eb68b4073099d3ed261f01f1385a4a9b37\\
& 1ea65efa8aee86ecdc01cd5e97467a789802e1fb3b4309cd459aeeaaa195\\
& 6b76efb1d958fb28482f98279747f2044d900b89d71560f93378481e0e1\\
& 9c24100ea2e236aac754847445cbf01115428892a124733be67a815125530\\
& 7413981 88b0ac8833682d5dceede3dc\\
s =& c8942cf2425b34f698553500a3e87ead8d87ef2c6a02e85868331fdb29a4\\
& 92e0
\end{aligned}
$$

有限小数から非負整数への変換

出力 $y = 0.934467$ は，プラスなので，$s_y = 0$ になる．小数部の桁数 b が 6 なので，変換ゲイン γ_y の下限条件は，$\gamma_y \geq 1 \times 10^6$ になる．ここで $\gamma_y = 1 \times 10^6$ にする．そうすると，$\gamma_y |y| = 934467$ になる．

同様に，比例ゲイン $K = -5.3$ に対して，$s_K = 1$ と $\gamma_K = 10$ にする．よって，$\gamma_K |K| = 53$ になる．

暗号化

関数 Enc() により，暗号化 $\mathsf{Enc}(\gamma_K|K|)$ と $\mathsf{Enc}(\gamma_y|y|)$ を行う．

$\gamma_K|K|$ 暗号文は $c_K = (c_{K1},\, c_{K2})$ とすると，以下のように得られる．ただし，乱数 $r_K = 866979022eeb07f4acb22ae33adfbce42e83a07032fdecf605d16905131706de$ である（十六進数）．

$$
\begin{aligned}
c_{K1} =& 3e89fd32cefcc042c7752e84b87745e5cd2996c0a0fccae281e09e660fac\\
& 4654900f17bbd18d4dcc87195f89600d5e17f96ecfdec7c3b25bab3ee9ca\\
& 018c693f6e0a8155f781ecd15914930f805b289e40f42a175bf67bc8900\\
& 60e7a549a08abf5e00ad9c5096d0c4b42e52a4236960aa6160c507fb6f5\\
& 89663fd137809710695b21c2431c6bd6ab8b1703796c8f88f60c3f14f2b\\
& e76a876167d492d1b91fd5ded14682893d10e2b1f8b5a9190952715592c\\
& 148023a8c2655410f22bec7dfa59c7dd18118a3030819b2c562bbd820a3c\\
& 683b4298b40c977c442d324097c93f01b543ec983f853bcaacef98fc0da8\\
& 2fd3f2a67c7d7662cd9ea93181d13a66fa28\\
c_{K2} =& 0c99427e3fbcd3b636627055a22ac69731283009341acfd107a9488e02a4\\
& 0dcfdacbdc478627aa7102fc6efabb2f834c642609912a2f4139db2e84b2\\
& aa42b66b5ad779196ed497fffef72dca6e39bff22efc384baa7805e26055\\
& 79397cbe94f56f4cce7e20021e8ed2c26d36a613f6b60814baf6d7b23a1c\\
& d2c9378d3581638d3813f6c7909528c5d1c5cedc417d81f834e7495421df\\
& 1109036e211336b7f7cfa9d8dbd9b49af16d25286011a03a26d4c371463\\
& f3dbd3b393daaac4e2e9b597ef5f612ab5b7b2f7b953328b5d904ada03db\\
& 0f3a80e1dc32f2a616963a0a2179d5df2eb337ed2738b0518fbb8a753a76\\
& dfb85ae946a38d1e1f2f42ae18be84329d
\end{aligned}
$$

同様に，$\gamma_y|y|$ の暗号文 $c_y = (c_{y1},\, c_{y2})$ が得られる．ただし，乱数 $r_y = cc659e50c84$

$2c7ab7c5172613c66a917835e1904dc6f531e99737334b7abc5fa$ である（十六進数）.

$$
\begin{aligned}
c_{y1} =& 2c1816b37f6628f7a00fe2263a21e19a5ec041850cf59cee5401a9c171b \\
& 3a65fc82fd139794fab148ea4121cb3075fca8ed295c15c44673a2ac3df \\
& 8a0ef744948a9dde1ac7c9e88d4ce3e6097e656fc4dfffa8a8f6aa12c0c \\
& ce17cd3096f70008709e1e1d63d2a17608a5bb0e00b567152bde94de699f \\
& 642b9cfea3367d4f8390ed1f972d8b186211ea5a30f81f26e0937f4a6c \\
& e5620aeb59b5de8800d32e2bb5a09fc3e54b10238c3eca0ee9b19416e857 \\
& cabe854312ce486b69bec8fbe4410ad1d91d6039cac5d5212aea358cb102 \\
& 4cbd7dc7a890b0c8e4d13e54db330120d6dca79e09ce7c5ddc1d9be3fd4a \\
& 57c7ffe0e55e1ed00baf4049225aea1fa075f \\
c_{y2} =& 700e2d0737f9c112c9e11eb6810be1c65ea6942a4976025f897ec23cfe18 \\
& fe1d36da4deb488151b8364b3e4dce9fb9ccff9d8ce71b0607f299551b31 \\
& c7067738b01a9f35a380debc75a77833d1de3f9ef1cce2e195b746ea80c4 \\
& 86fc3d24a84c3fc7f544595e824e0190c419e2a49e6f115e452e2442736 \\
& 58d44c95487f5a18701032ab74be8fdf66ffc6c6cd7fbc9d75e9f931964 \\
& 23717336067d414339e53d61a7eb56b626bb507bf3d6c7240e15623e7e5c \\
& 1f74feef3eea698f5016c36476cb259ab3fcf4a97be47a5832255273841 \\
& 92090d2a2542ef5e363f665a5ef8b051d3ac6e07648d49e1036deea42b4 \\
& 344da679d9d3011a9c5c6a4630e32c56f1e4
\end{aligned}
$$

暗号文同士の演算

関数 Mul_C() により，暗号文同士の演算 $\mathsf{Enc}(\gamma_K|K|) * \mathsf{Enc}(\gamma_y|y|)$ を行う．結果の暗号文を $c_u = (c_{u1},\, c_{u2})$ とすると，$(c_{u1},\, c_{u2}) = (c_{K1}c_{y1},\, c_{K2}c_{y2})$ が得られる．

$$
\begin{aligned}
c_{u1} =& 581d9452f1d3a09aa72029e7b5a1ce3d5557ed9cec725e97684e33998b8\\
& 73cd7ef174b3ab0ccaae7432ce630b1f4d34ed0c3cf8e3cb7ebe04b3e143\\
& a39d62eb8c7c2e50a9cdf48e3887543056221ee6315fe229ecfc6287f508\\
& 599383db16d0e7df777cadd4906c06c94512350b205684019ba4484d7526\\
& f99804a36716a6d6f728f164fda8681d24572b8fa6775074972fe9bfb2\\
& 0d5b4396102151b2ec54f875e1e3f48055d38097408fb083acc2cb02ba8\\
& 9147dbf0ee1bb7b221c786768b3b3552548ac676fff67503705ee5ca446f2\\
& a8f602e560453c0f31fafcd6ca5ac20915d78f78559a2348cc162890556\\
& aef9ebd640fa7772999a23b35157a247c4277\\
c_{u2} =& 9499febb2f77d7153c77b09f6151cca129a2cb59a33755193890f10bd31d\\
& c94e14891b995369fff88c1d4b7085f2a34092c4a16d7f512b3f8fbc31fc\\
& 30fff2dacca9801fcfac6eda63773e296070a3736266b6d957817e5e39f1\\
& e67c042593dadf824dae5f67ce104a489a111a09e4b2320367d766ca17ab\\
& be3f53a1a92bb96c99c111eca7831a77336ed328b406be41dbd043125b42b\\
& 7b5e08c3227e15bea348fa121566678469a7a677d36082a88f0006f88ff\\
& 4bbc72561a8dc7af537029610c8cf86c8d9e1cc48ee4716f595c0992c5221\\
& e12fcea4e44bf70430d3a5671c6ae011ee9950bf14a05764f59b5d7a1f0e\\
& 488646b2b4448d1ab149d21e35d1abe3d
\end{aligned}
$$

復号化

関数 Dec() により，復号化すると，

$$\mathsf{Dec}(\mathsf{Enc}(\gamma_K|K|) * \mathsf{Enc}(\gamma_y|y|)) = 49526751$$

が得られる．

非負整数から有限小数への逆変換

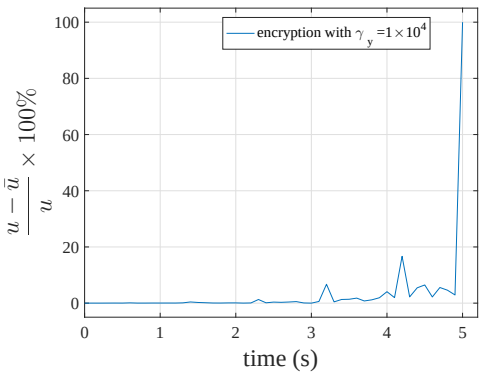
$\bar{u}$ は次のように得られる．

$$
\begin{aligned}
& s_K \oplus s_y = 1 \oplus 0 = 1\\
& \bar{u} = (-1)^{s_K \oplus s_y} \mathsf{Dec}(\mathsf{Enc}(\gamma_K|K|) * \mathsf{Enc}(\gamma_y|y|))/(\gamma_K\gamma_y) = -4.9526751
\end{aligned}
$$

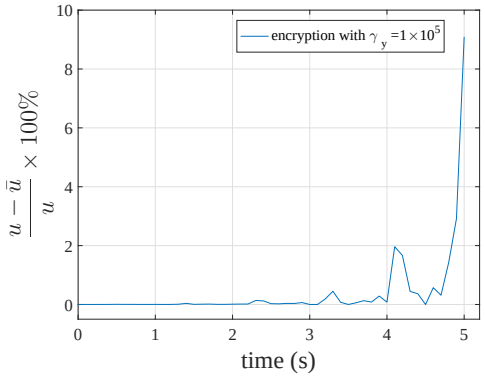
同様に，他の $\bar{u}(t)$ を求め，Table 4.2 に示す．同表により，暗号化する場合の制御入力 $\bar{u}(t)$ は暗号化しない場合の制御入力 $u(t)$ と一致することがわかる．よって，提案手法による誤差除去の有効性が確認された．

最後に，$\gamma_y \geq 1 \times 10^6$ を例として，変換ゲインの下限条件について確認する．

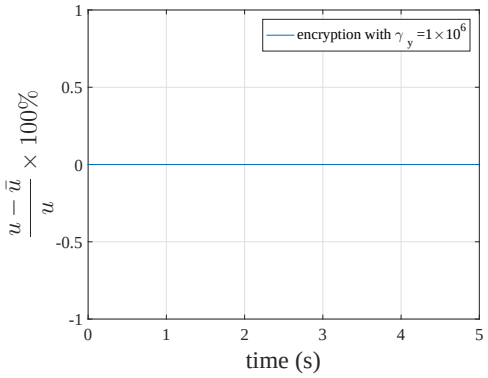
$\gamma_K = 10$ を変えず，γ_y を，1×10^4，1×10^5 と 1×10^7 に逐次変え，得られた $\bar{u}(t)$ を，付録 C に示す．相対誤差 $\frac{u-\bar{u}}{u} \times 100\%$ を計算し，Fig. 4.2 に示す．そして，Table 4.3 に相対誤差の最大値を示す．Table 4.3 により，誤差は $\gamma_y = 1 \times 10^4$ の場合，相対誤差の最大値は 100%，$\gamma_y = 1 \times 10^5$ の場合，相対誤差の最大値は 9.1%，$\gamma_y = 1 \times 10^6$ と $\gamma_y = 1 \times 10^7$ の場合，0 になる．



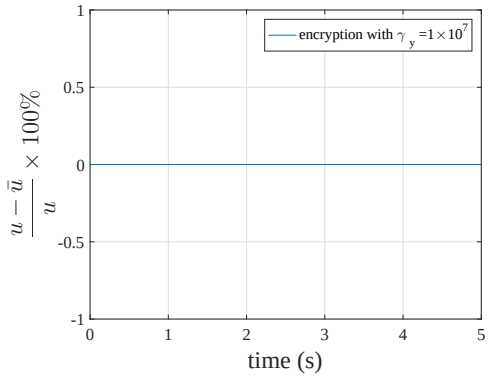
(a) $\gamma_y = 1 \times 10^4$



(b) $\gamma_y = 1 \times 10^5$



(c) $\gamma_y = 1 \times 10^6$



(d) $\gamma_y = 1 \times 10^7$

Fig. 4.2: 相対誤差 $\frac{u-\bar{u}}{u} \times 100\%$

Table 4.2: $\gamma_K = 10, \gamma_y = 1 \times 10^6$ の場合の $\bar{u}(t)$

t(s)	$u(t)$	$\bar{u}(t)$	t(s)	$u(t)$	$\bar{u}(t)$
0	-5.3	-5.3	2.6	0.0988662	0.0988662
0.1	-4.9526751	-4.9526751	2.7	0.1026981	0.1026981
0.2	-4.0753502	-4.0753502	2.8	0.0927288	0.0927288
0.3	-2.9128694	-2.9128694	2.9	0.0737177	0.0737177
0.4	-1.6832111	-1.6832111	3.0	0.05035	0.05035
0.5	-0.5571148	-0.5571148	3.1	0.026659	0.026659
0.6	0.3511886	0.3511886	3.2	0.0056816	0.0056816
0.7	0.9838231	0.9838231	3.3	-0.0106477	-0.0106477
0.8	1.3321232	1.3321232	3.4	-0.0214809	-0.0214809
0.9	1.4251382	1.4251382	3.5	-0.026871	-0.026871
1.0	1.3163981	1.3163981	3.6	-0.0275229	-0.0275229
1.1	1.0709816	1.0709816	3.7	-0.0245708	-0.0245708
1.2	0.7544179	0.7544179	3.8	-0.0193079	-0.0193079
1.3	0.4244081	0.4244081	3.9	-0.0129691	-0.0129691
1.4	0.1256047	0.1256047	4.0	-0.0066303	-0.0066303
1.5	-0.1126303	-0.1126303	4.1	-0.0010812	-0.0010812
1.6	-0.2759975	-0.2759975	4.2	0.00318	0.00318
1.7	-0.3632037	-0.3632037	4.3	0.0059625	0.0059625
1.8	-0.3827819	-0.3827819	4.4	0.0072875	0.0072875
1.9	-0.3495509	-0.3495509	4.5	0.007367	0.007367
2.0	-0.2811385	-0.2811385	4.6	0.0065031	0.0065031
2.1	-0.1950665	-0.1950665	4.7	0.0050509	0.0050509
2.2	-0.1065989	-0.1065989	4.8	0.0033337	0.0033337
2.3	-0.0273851	-0.0273851	4.9	0.0016377	0.0016377
2.4	0.0350224	0.0350224	5.0	0.0001749	0.0001749
2.5	0.0771362	0.0771362			

Table 4.3: 相対誤差の最大値

γ_y	1×10^4	1×10^5	1×10^6	1×10^7
相対誤差の最大値 (%)	100	9.1	0	0

4.3　計算時間

制御システムは周期的かつリアルタイムに制御処理が行われているので，暗号化する場合の計算時間を調べる必要がある．また，ElGamal 暗号において，全ての演算ではモジュロ p（mod p）をとり，暗号化の際，ランダムな乱数 r を用い，そして，秘密鍵 s により復号化を行う．よって，p，r と s のサイズが計算時間に影響を与えると予想される．

そこで，本節では，p，r と s のサイズごとの計算時間について調べる．計算時間計測環境を Table 4.4 に示す．C++の chorono により計算時間を計測する．

Table 4.4: 実行環境

CPU	2.7GHz Intel Core i5
memory	8GB 1600 MHz DDR3
OS	macOS Mojave ver.10.14
Language	C++(Xcode ver. 10.1)

まず，公開鍵 p のサイズごとの計算時間について調べる．サイズが 128，256，512，1024，2048 ビットである p を生成し，付録 B に示す．同付録により，p が長いほど，生成時間が増える．

乱数 r と秘密鍵 s とのサイズを 30 ビット，変換ゲインを $\gamma_K = 10$ と $\gamma_y = 1 \times 10^6$ とする．上記の 128 から 2048 ビットまでの 5 通りの鍵を用い，0s から 5s までの制御入力 $\bar{u}(t)$ を 51 個求めて，計算時間を記録する．時間の平均値を Table 4.5 に示す．

同表より，非負整数と有限小数との間の処理の計算時間 t_1 と t_5 は，p から影響を受けていないが，他の各処理段階の計算時間は p の増大に伴い伸びることがわかる．

Table 4.5: 公開鍵 p ごとの平均計算時間（ms）

p(bit)	128	256	512	1024	2048
非負整数への変換 t_1	0.001	0.001	0.001	0.001	0.001
暗号化 t_2	1.8	4	16	52	142
暗号文同士の演算 t_3	0.047	0.109	0.37	1.24	3.32
復号化 t_4	2	6	20	69	210
有限小数への逆変換 t_5	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
合計 (四捨五入)t	4	10	36	122	355

次に，乱数 r のサイズごとの計算時間について調べる．公開鍵 p のサイズを 2048 ビット，秘密鍵 s のサイズを 250 ビット，変換ゲインを $\gamma_K = 10$ と $\gamma_y = 1 \times 10^6$ とする．r のサイズを 30，100 と 250 ビットにした場合の計算時間の平均値を Table 4.6 に示す．同表により，乱数 r が長いほど，暗号化する時間 t_2 が伸びるが，他の計算時間はほぼ変化しないことがわかる．

Table 4.6: 乱数 r ごとの平均計算時間（ms）

r（bit）	30	100	250
非負整数への変換 t_1	0.001	0.001	0.001
暗号化 t_2	138	541	1430
暗号文同士の演算 t_3	3.327	3.372	3.472
復号化 t_4	872	850	864
有限小数への逆変換 t_5	1×10^{-4}	1×10^{-4}	1×10^{-4}
合計 (四捨五入)t	1013	1394	2297

最後に，秘密鍵 s のサイズごとの計算時間について調べる．公開鍵 p のサイズを 2048 ビット，乱数 r のサイズを 250 ビット，変換ゲインを $\gamma_K = 10$ と $\gamma_y = 1 \times 10^6$ とする．s のサイズを 30，100，250 ビットにした場合の計算時間の平均値を Table 4.7 に示す．同表により，乱数 r が長いほど，復号化する時間 t_4 が伸びることが，他の各計算時間はほぼ変化しないことがわかる．

Table 4.7: 秘密鍵 s ごとの平均処理時間（ms）

s（bit）	30	100	250
非負整数への変換 t_1	0.001	0.001	0.001
暗号化 t_2	1441	1426	1430
暗号文同士の演算 t_3	3.333	3.420	3.472
復号化 t_4	209	411	864
有限小数への逆変換 t_5	1×10^{-4}	1×10^{-4}	1×10^{-4}
合計 (四捨五入)t	1653	1840	2297

第5章　おわりに

本章では，本論文のまとめと今後の課題を述べる．

5.1　まとめ

本論文では，盗聴の可能性があるネットワーク化制御系において，有限小数値の信号やパラメータしか持たない比例制御器に対して，原始根に基づく ElGamal 暗号を用い，内部情報を暗号化したまま元の制御入力を求める手法を提案した．はじめに，原始根に基づく ElGamal 暗号の原理や理解に必要な数学の基礎知識を解説した．次に，制御器が盗聴を受ける状態で，制御器内部情報を暗号化したまま元の制御入力を求める手法を提案した．提案手法が有効になるために変換ゲインの満たすべきの条件を示した．最後に，2048 ビットの鍵長の ElGamal 暗号を用いる数値例により，提案手法の有効性及び変換ゲインの下限条件を検証した．さらに，各処理段階の計算時間を調べた．本提案手法は，誤差を完全に除去することができるが，制御器の内部信号の符号の情報は保護できない．

5.2　今後の課題

本研究の今後の課題を以下に三点挙げる．一点目は，長い鍵の生成プログラムの開発である．本論文で用いる 2048 ビットの鍵長の ElGamal 暗号は，現在は安全であると予想されているが，コンピュータの能力や解読技術によって，将来は安全のためにもっと長い鍵長が必要だと思われる．しかし，本論文での鍵生成プログラムでは，2048 ビットまでの鍵しか生成できなかった．2048 ビット以上の鍵長を効率よく生成するプログラムを開発する必要がある．二点目は，計算時間を短くすることである．しかし， 制御システムは周期的かつリアルタイムに制御処理が行われている．その周期は数 msec から 1000msec 程度である．特に，数 msec から数十 msec 単位での周期的な制御処理が求められる場合も多い．しかし，本論文で，2048 ビットの鍵長の場合，計算時間は 1000msec や 2000msec まで伸びる．そして，実機の場合，通信時間や制御対象の動作時間を加えると，もっと長く時間がかかると予想される．制御系のリアルタイム処理への対応が難しいと思われる．

今後は計算処理を高速化する必要がある．最後に本論文では，単入力単出力の比例制御に対して，誤差を除去する手法を提案したが，これをもとに，線形制御やモデル予測制御などのもっと複雑な制御系に対して，誤差を除去する手法を検討することが考えられる．

参考文献

1) 新誠一. 社会インフラへのサイバー攻撃に対する課題と取り組み. 情報処理, Vol. 55, No. 7, pp. 640–646, 2014.

2) 情報処理推進機構. 重要インフラの制御システムセキュリティーと it サービス接続に関する調査. https://www.ipa.go.jp/files/000013981.pdf, 3 2009.

3) 楠正憲. 頻発するサイバー攻撃被害:組織にとっての課題と対策. 情報管理, Vol. 59, No. 9, pp. 599–606, 2016.

4) ロイター. ホンダ、狭山工場の操業を一時停止　サーバー攻撃でウイルス汚染. https://jp.reuters.com/article/honda-factory-virus-idJPKBN19C0M3, 2017.

5) H. Nishion and H. Ishii. Distributed detection of cyber attacks and faults for power systems. In *Proceedings of 19th World Congress The International Federation of Automatic Control*, pp. 11932–11937, 2014.

6) Q. Zhu and T. Basar. Robust and resilent control design for cyber-physical systems with an application to power systems. In *Proceedings of 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 4066–4071, 2011.

7) Z. Pang, G. Zheng, G. Liu, and C. Luo. Secure transmission mechanism for networked control systems under deception attacks. In *Proceedings of the 2011 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*, pp. 27–32, 2011.

8) 伯田ら. 特集制御システムセキュリティの現状と課題：制御用コントローラ向け暗号通信機能の実現に向けて. 計測と制御, Vol. 53, No. 10, pp. 936–942, 2014.

9) K. Kogiso and T. Fujita. Cyber-security enhancement of networked control systems using homomorphic encryption. In *Proceedings of IEEE Conference on Decision and Control*, pp. 6836–6843, 2015.

10) 馬場, 日下, 小木曽. オブザーバを用いた暗号化制御系および実機検証. 第 60 回自動制御連合講演会, 2017.

11) 日下, 小木曽. 多倍長整数ライブラリを用いた暗号化制御系におけるリアルタイム性の実験的考察. 第 61 回自動制御連合講演会, 2018.

12) K. Kogiso. Upper-bound analysis of performacne degradation in encrypted control system. In *American Control Conference*, pp. 1250–1255, 2018.

13) F. Farokhi, I. Shames, and N. Batterham. secure and private control using semi-homomorphic encryption. *Control Engineering Practice*, Vol. 67, pp. 13–20, 2017.

14) Y. Shoukry, K. Gatsis, A. Alanwar, G. J. Pappas, S. A. Seshia, M. Srivastava, and P. Tabuada. Privacy-aware quadratic optimization using partially homomorphic encryption. In *IEEE Conference on Decision and Control*, pp. 3576–3581, 2016.

15) J. Kim, C. Lee, H. Shim, J. H. Cheon, A. Kim, M. Kim, and Y. Song. Encrypting controller using fully homomorphic encryption for security of cyber-physical systems. In *IFAC Workshop on Distributed Estimation and Control in Networked Systems*, Vol. 49, pp. 175–180, 2016.

16) A. B. Alexandru, K. Gatsis, and G. J. Pappas. Privacy preserving cloud-based quadratic optimization. In *Fifty-Fifth Annual Allerton Conference*, pp. 1168–1175, 2017.

17) T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO'84*, Vol. 196, pp. 10–18, 1984.

18) B. Elaine, B. William, B.William, P.William, and S.Miles. Recommendation for key management part 1: General (revision 3). *NIST Special Publication 800-57*, 2012.

19) 楫元. 工科系のための初等整数論入門 : 公開鍵暗号をめざして. 培風館, 2000.

20) IPUSIRON. 暗号技術のすべて. 翔泳社, 2017.

21) 森山大輔, 西巻陵, 岡本龍明. 公開鍵暗号の数理. 共立出版, 2011.

22) 柴田望洋. 新・明解 C++入門. SB クリエイティブ, 2017.

23) 橋本晋之介. RSA : 暗号技術の基礎から C++による実装まで. ソフトバンクパブリッシング, 2001.

24) 野下浩平ら. 基本的算法. 岩波書店, 1983.

25) 池野信一, 小山謙二. 現代暗号理論. 電子通信学会, 1986.

26) 岡本栄司. 暗号理論入門. 共立出版, 1993.

付録A　ソースコード

本論文でのソースコードは，文献 [23] を参考にするものである．プログラムを分かりやすくためエラー処理を除いている．まず，プログラムの実現方針を述べ，ソースコードを示す．

ElGamal 暗号をソフトウエアで実現するための必要な機能は以下の通りである．

- 長大な整数の表現
- 数値演算
- 逆数の求め
- 素数判定と乱数生成
- 原始根判定

実行速度を優先とするので，プログラムは C++ [22] で組むことにする．上記の機能を実現する方針は以下である．

長大な整数の表現

まず，ElGamal 暗号では，512 ビットや 1024 ビットなどといった長大な整数を扱わなければならない．しかし，通常の C++の long 型では，64 ビットまでの整数しか表現できないので，64 ビット以上の整数なら，ソフトウェアで表現する必要がある．本研究では，配列で数を扱う．配列要素を数の桁，そして，配列要素の 0 番目を最下位の桁とし，次のようなルールでソフトウェアを組む [23]．

1. short 型が 16 ビット，int 型が 32 ビットのコンパイラとする
2. unsigned short 型の値を数の 1 桁とする
3. unsigned int 型の値を中間結果として使用する

この場合，基数は 65536(16 ビット，unsigned short 型の大きさ) となり，数は 65536 進数で表現する．ソフトウェアでは C++のクラスを用いて配列をまとめて管理する．

数値演算

ElGamal 暗号では，処理のほとんどは数値演算で，その内部は四則演算などの基本的な演算と判定の繰り返しである．ここで必要な演算には，次のようなものがある．

- 四則演算，シフト演算，比較演算，代入演算，剰余算，べき乗算

日常では 10 進数を使い，筆算するときは 1 桁ずつ行う．この仕組みを真似て，65536 進数で 1 桁ずつ計算を行うというように設計する．

除算の原理について文献[24]を参考されたい．べき乗算には，反複平方積による指数計算法というアルゴリズム[25]が採用される．これは指数を因数分解し法をとりながら積を計算していく方法である．この方法を用いると，中間結果があまり大きくならない上に，乗算の回数も少なく済む．

逆数の求め方

法 n，整数 a に対して，$ab = 1 \mod p$ となる整数 b を a の逆数と呼ぶ．この逆数を求めるために，拡張ユークリッドの互除法[25, 26]を使用する．

素数判定と乱数生成

ElGamal 暗号では公開鍵とする法 p が素数なので，素数生成が必要になる．しかし，素数を直接作り出すアルゴリズムが発見されていない．そこで，ある数が素数であるかどうかを確かめる素数判定法のアルゴリズムが考案されている．

本研究では汎用的なラビン法[26]を採用する．ラビン法は，Rabin が定案した，高い確率で素数を判定できる確率的素数判定法である．乱数生成には標準ライブラリの乱数生成関数 rand() が使用される．

原始根判定

原始元を効率よく生成するために，次のような形式の素数 p が採用される．

$$p = 2ab + 1$$

ここで a と b は素数である．$p - 1 = 2ab$ において，素因数は 2，a と b しかないので，原始元判定法の検証は，3 回だけで済む．

$p - 1$ の素因数が全て小さい場合に，離散対数を効率的に計算するポーリッヒ・ヘルマン (Pohig-Hellman) というアルゴリズムが存在するため，ElGamal 暗号の鍵生成アルゴリズムにおいて，$p - 1$ が大きな素因数を含むという条件が望ましい．

Listing A.1: source code

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <ctype.h>
6  #include <time.h>
7  #include <iomanip>
8  using namespace std;
9  const int COL_LENGTH =16;
10 const int COL_MASK = 0xffff;
11 const int BASE_UNIT = 0x10000;
12 const int HALF_UNIT = 0x8000;
13 const int DEF_COLS = 4;
14 typedef unsigned short u_short;
15 typedef unsigned int u_int;
16 typedef unsigned long u_long;
17 class MpI{
18 public:
19     u_short* v;
20     int l;
21 public:
22     MpI();
23     MpI( const int );
24     MpI( const u_long );
25     MpI( const int , const int );
26     MpI( const char* );
27     MpI( const MpI& );
28     ~MpI();
29 public:
30     void print() const;
31     int getBitLength() const;
32     int getMaxColumn() const;
33     int changeLength( int );
34     void adjustLength();
35     MpI operator++();
36     MpI operator++( int );
37     MpI operator--();
38     MpI operator--( int );
39     MpI& operator=( const MpI& );
40     MpI& operator+=( const MpI& );
41     MpI& operator-=( const MpI& );
42     MpI& operator*=( const MpI& );
43     MpI& operator%=( const MpI& );
44     MpI& operator>>=( const int );
45     MpI& operator<<=( const int );
```

```
46 };
47 MpI operator+( const MpI&, const MpI& );
48 MpI operator-( const MpI&, const MpI& );
49 MpI operator*( const MpI&, const MpI& );
50 MpI operator/( const MpI&, const MpI& );
51 MpI operator%( const MpI&, const MpI& );
52 MpI div( const MpI&, const MpI&, MpI& );
53 u_short div2( const u_short a1, const u_short a2, const u_short b
       );
54 MpI operator>>( const MpI&, const int );
55 MpI operator<<( const MpI&, const int );
56 int operator==( const MpI&, const MpI& );
57 int operator!=( const MpI&, const MpI& );
58 int operator>=( const MpI&, const MpI& );
59 int operator<=( const MpI&, const MpI& );
60 int operator>( const MpI&, const MpI& );
61 int operator<( const MpI&, const MpI& );
62 int isEven(const MpI& );
63 MpI exp(const MpI&, const MpI&, const MpI& );
64 MpI random_void( );
65 MpI random( const int );
66 MpI randomBit( const int );
67 int isprime( const MpI& );
68 MpI inv(const MpI&,const MpI& );
69 void randInitial();
70 MpI PrimeGen( const int );
71 void Key_Gen(MpI&, MpI&, MpI&, MpI&, const int& );
72 void Enc( MpI&, MpI&, MpI&, MpI&, MpI&, MpI& );
73 MpI mul_mod(const MpI& a, const MpI& b, const MpI& n );
74 void Mul_C(MpI& c_1, MpI& c_2, const MpI& c1_1, const MpI& c1_2,
75              const MpI& c2_1, const MpI& c2_2, const MpI& p);
76 MpI Dec(MpI& c_2, MpI& c_1, MpI& s, MpI& p);
77 int main() {
78     double y=0.934467,K = -5.3, u_;
79     u_int sK, sy;
80     u_long u_t, rK=10, ry=1e6;
81     MpI m1, m2, m, m_, cK_1, cK_2, cy_1, cy_2, cu_1, cu_2;
82
83     randInitial();
84     //key generation
85     //Key_Gen(p, g, s, h, k);
86     MpI p="a81a02f3da153e9167455930977d7bd830251137e5c5
87     c184567d1219dacfe9c057da3d8aa4bbc7e1a67a15e4
88     818ee1085b94451e8cb3de9cd50cd49f68cda2aaa378
89     c72e673de14c9fb054eba487e567bf09af765f6379e4
90     983dd01416e338994ba5f21225676a83f2f2c7551d49
```

```
91      606e6fef9674a2c20c951cc9e3fb7fb04dd132ef15df
92      c32ebdd1e5b6a614ce64cf2fdab144925c8010a7ac7d3
93      0f5d02df4f1b4db902e8240b97c4fcbe0498c386eab1a
94      bc63e39dc0e33668ad76b7aee751e79306a3bf56a68c1
95      b3eadeab2c6e8472f5384f59dcd9c70be74b728f8040f
96      5862e188c8c4783456eb386e98a70586e4ad312ed3a95b3
97      5c86aa649d9ab42d07e77";
98      MpI g = 2;
99      MpI s ="c8942cf2425b34f698553500a3e87ead8d87e
100     f2c6a02e85868331fdb29a492e0";
101     MpI h;
102     h = exp(g, s, p);h.print();
103
104     //conversion to nonnegative integer
105     if (K < 0) {sK = 1; m1 = u_long(-K * rK);}
106     else{ sK = 0; m1 = u_long(K * rK);}
107     if (y < 0) { sy = 1; m2 = u_long(-y * ry);}
108     else{ sy = 0; m2 = u_long(y * ry);}
109
110     //encryption
111     Enc(m1, cK_1, cK_2, g, h, p);
112     Enc(m2, cy_1, cy_2, g, h, p);
113
114     //multiplication of cipher
115     Mul_C(cu_1, cu_2, cK_1, cK_2, cy_1, cy_2, p);
116
117     //decryption
118     m_ = Dec(cu_2, cu_1, s, p);
119
120     // conversion to decimal
121     u_t = m_.v[3]*0x1000000000000 + m_.v[2]*0x100000000 + m_.v
            [1]*BASE_UNIT + m_.v[0];
122     if( sK == sy ){
123         u_ = u_t / (rK * ry * (double)1.0);
124      }else{
125         u_= u_t / (rK * ry *(double)-1.0);
126      }
127
128     cout << setprecision(10) << u_<< '\n';
129     return 0;
130 }
131 MpI::MpI(){
132     l = DEF_COLS; v = new u_short[l];
133     for(int i=0; i<l; i++) v[i] = 0;
134 }
135 MpI::MpI(const int val ){
```

```
136     l = DEF_COLS; v = new u_short[l];
137     for(int i=2; i<DEF_COLS; i++) v[i] = 0;
138     v[0] = u_short( val & COL_MASK );
139     v[1] = u_short( val >> COL_LENGTH );
140 }
141 MpI::MpI(const u_long val ){
142     u_long t = val;
143     l = DEF_COLS; v = new u_short[l];
144     v[0] = u_short( t & COL_MASK );
145     v[1] = u_short( t >>= COL_LENGTH );
146     v[2] = u_short( t >>= COL_LENGTH );
147     v[3] = u_short( t >>= COL_LENGTH );
148 }
149 MpI::MpI(const int size, const int val ){
150     l = (((size - 1) / DEF_COLS) + 1 ) * DEF_COLS;
151     v = new u_short[l] ;
152     for(int i=0; i<l; i++) v[i] = u_short( val & COL_MASK );
153 }
154 MpI::MpI( const char* str ){
155     int strL = (int)strlen( str );
156     l = strL/4 + ((strL%4)?1:0);
157     l = (((l - 1) / DEF_COLS) + 1 ) * DEF_COLS;
158     v = new u_short[l];
159     for( int i=0; i<l; i++ ){
160         char h[5]; h[4] = '\0';
161         for( int j=3; j>=0; j-- ){
162             if( strL > 0 ){ h[j] = str[--strL];}
163             else h[j] = '0';
164         }
165         v[i] = u_short( 0xffff & strtol( h, NULL, 16 ) );
166     }
167 }
168 MpI::MpI(const MpI& a ){
169     l = a.l; v = new u_short[l];
170     for (int i=0; i<l; i++) v[i] = a.v[i];
171 }
172 MpI::~MpI( ){ delete[] v;}
173 inline void MpI::print() const{
174     for( int i=l-1; i>=0; i-- ){
175         cout.setf(ios::hex,ios::basefield);
176         cout.width(4);
177         cout.fill('0');
178         cout << v[i];
179     }
180     cout << '\n';
181 }
```

```
182 inline int MpI::getBitLength() const{
183     for( int i=l-1; i>=0; i-- ){
184         if( v[i] != 0 ){
185             int bits = 0; u_short t = v[i];
186             do{ bits++; t >>= 1;}while( t != 0 );
187             return bits + i * COL_LENGTH;
188         }
189     }
190     return 0;
191 }
192 inline int MpI::getMaxColumn() const{
193     int i;
194     for(i=l-1; i>=0; i--) if( v[i] != 0 ) break;
195     return i;
196 }
197 inline int MpI::changeLength( int newLength ){
198     int nl = (((newLength - 1) / DEF_COLS) + 1 ) * DEF_COLS;
199     if( l == nl ) return 0;
200     if( l < nl ){
201         u_short* p = new u_short[nl] ;
202         int i=0;
203         for( ; i<l; i++ ) p[i] = v[i];
204         for( ; i<nl; i++ ) p[i] = 0;
205         delete[] v;
206         l = nl; v = p;
207         return 0;
208     }else{
209         u_short* p = new u_short[nl] ;
210         for( int i=0; i<nl; i++ ) p[i] = v[i];
211         delete[] v;
212         l = nl; v = p;
213         return 0;
214     }
215 }
216 inline void MpI::adjustLength( void ){ changeLength( getMaxColumn
        () + 1 );}
217 inline MpI& MpI::operator=( const MpI& a ){
218     if( l && v ) delete[] v;
219     l = a.l; v = new u_short[l] ;
220     for( int i=0; i<l; i++ ) v[i] = a.v[i];
221     return *this;
222 }
223 inline MpI operator+( const MpI& a, const MpI& b ){
224     int ll, ls;
225     u_short* p;
226     if( a.l > b.l ){
```

```
227         ll = a.l; ls = b.l; p = a.v;
228     } else{
229         ll = b.l; ls = a.l; p = b.v;
230     }
231     MpI z( ll+1, 0 );
232     int i, c = 0, t;
233     for( i=0; i<ls; i++ ){
234         t = a.v[i] + b.v[i] + c;
235         z.v[i] = u_short( t );
236         c = t >> COL_LENGTH;
237     }
238     for( ; i<ll; i++ ){
239         t = p[i] + c;
240         z.v[i] = u_short( t );
241         c = t >> COL_LENGTH;
242     }
243     z.v[ll] = u_short( c );
244     return z;
245 }
246 inline MpI operator-( const MpI& a, const MpI& b ){
247     MpI z( a.l, 0 );
248     int ls = b.getMaxColumn();
249     int i=0, c = 0, t;
250     for( ; i<=ls; i++ ){
251         t = int( b.v[i] ) + c;
252         if( int(a.v[i]) < t ){
253             z.v[i] = u_short( BASE_UNIT + int(a.v[i]) - t );
254             c = 1;
255         }else{
256             z.v[i] = u_short( a.v[i] - t );
257             c = 0;
258         }
259     }
260     for( ; i<a.l; i++ ){
261         if( a.v[i] < c ){
262             z.v[i] = u_short( BASE_UNIT + int(a.v[i]) - c );
263             c = 1;
264         }else{
265             z.v[i] = u_short( a.v[i] - c );
266             c = 0;
267         }
268     }
269     return z;
270 }
271 inline MpI operator*( const MpI& a, const MpI& b ){
272     int ll = a.l + b.l;
```

```
273     MpI d( ll, 0 );
274     int i = 0, j = 0;
275     for( ; i<b.l; i++ ){
276         u_short k = 0;
277         for( j=0; j<a.l; j++ ){
278             int tmp = a.v[j] * b.v[i] + d.v[i+j] + k;
279             d.v[i+j] = u_short( tmp & COL_MASK );
280             k = u_short( tmp >> COL_LENGTH );
281         }
282         d.v[i+j] = k;
283     }
284     d.adjustLength();
285     return d;
286 }
287 inline MpI operator/( const MpI& a, const MpI& b ){
288     MpI r; return div( a, b, r );
289 }
290 inline MpI operator%( const MpI& a, const MpI& b ){
291     MpI r; div( a, b, r ); return r;
292 }
293 inline u_short div2( const u_short a1, const u_short a2, const
        u_short b ){
294     u_int aa = a1 * BASE_UNIT + a2;
295     u_int q = aa / b;
296     if( q >= BASE_UNIT ) q = BASE_UNIT-1;
297     u_int t1 = q * b;
298     while( t1 > aa ){ q = q - 1; t1 -= b; }
299     return (u_short)q;
300 }
301 inline MpI div( const MpI& a, const MpI& b, MpI& r ){
302     MpI q( a.l, 0 ); r = 0;
303     if( a < b ){ r = a; return q; }
304     int bp = b.getMaxColumn();
305     MpI opa = a, opb = b;
306     while( opb.v[bp] < HALF_UNIT ){
307         opb <<= 1; opa <<= 1;
308     }
309     int ap = opa.getMaxColumn();
310     if( ap < 1 ){
311         opa <<= COL_LENGTH; opb <<= COL_LENGTH;
312         ap++; bp++;
313     }
314     int f = 0;
315     if( ap == bp ){ opa <<= COL_LENGTH; f = COL_LENGTH; ap++;}
316     MpI bb = opb << (ap-bp)* COL_LENGTH;
317     if( opa >= bb ){
```

```
318         q.v[ap-bp] = 1; opa -= bb; ap = opa.getMaxColumn();
319     }
320     u_short op1 = opa.v[ap]; u_short op2 = opa.v[ap-1];
321     MpI tmpa, tmpr;
322     for( int i=ap-bp-1; i>=0; i-- ){
323         q.v[i] = div2( op1, op2, opb.v[bp] );
324         tmpa = (q.v[i] * opb) << (COL_LENGTH * i);
325         while( opa < tmpa ){
326             if( q.v[i] > 0 ) q.v[i]--;
327             else{ MpI s( q.l, 0 ); s.v[i] = 1; q -= s;}
328             tmpa -= (opb << (COL_LENGTH * i));
329         }
330         tmpr = opa - tmpa;
331         if( tmpr < opb ) break;
332         opa = tmpr;
333     loopDIV:
334         ap--;
335         if( ap < 1 ) break;
336         op1 = opa.v[ap]; op2 = opa.v[ap-1];
337         if( op1 == 0 && op2 < opb.v[bp] ){
338             if( --i > 0 ){ q.v[i] = 0; goto loopDIV;}
339             else break;
340         }
341     }
342     q >>= f; q.adjustLength(); r = a - q * b; return q;
343 }
344 inline MpI MpI::operator++(){ *this = *this + 1; return *this; }
345 inline MpI MpI::operator++( int n )
346 { MpI r = *this; *this = *this + 1; return r; }
347 inline MpI MpI::operator--(){ *this = *this - 1; return *this; }
348 inline MpI MpI::operator--( int n )
349 { MpI r = *this; *this = *this - 1; return r; }
350 inline MpI operator>>( const MpI& a, const int n ){
351     if( n < 0 ) return a<<(-n);
352     MpI r = a;
353     if( n == 0 ) return r;
354     if( n >= COL_LENGTH * r.l ) return 0;
355     int w = n / COL_LENGTH;
356     int b = n - w * COL_LENGTH;
357     int i;
358     for( i=0; i<r.l-1-w; i++ )
359         r.v[i] = u_short(( (r.v[i+w+1] << (COL_LENGTH-b)) | (r.v
                [i+w] >> b) ) & COL_MASK );
360     r.v[i] = u_short(( r.v[i+w] >> b ) & COL_MASK );
361     for( i++; i<r.l; i++ ) r.v[i] = 0;
362     r.adjustLength(); return r;}
```

```
363 inline MpI operator<<( const MpI& a, const int n ){
364     if( n < 0 ) return a>>(-n);
365     MpI r = a;
366     if( n == 0 ) return r;
367     int w = n / COL_LENGTH;
368     int b = n - w * COL_LENGTH;
369     r.changeLength( w + 1 + r.l );
370     int i;
371     for( i=r.l-1; i>w; i-- )
372         r.v[i] = u_short( ( (r.v[i-w-1] >> (COL_LENGTH-b)) | (r.
                v[i-w] << b) ) & COL_MASK );
373     r.v[i] = u_short( ( r.v[i-w] << b ) & COL_MASK );
374     for( i--; i>=0; i-- ) r.v[i] = 0;
375     r.adjustLength(); return r;}
376 inline MpI& MpI::operator+=( const MpI& a ){ *this = *this + a;
       return *this; }
377 inline MpI& MpI::operator-=( const MpI& a ){ *this = *this - a;
       return *this; }
378 inline MpI& MpI::operator*=( const MpI& a ){ *this = *this * a;
       return *this; }
379 inline MpI& MpI::operator%=( const MpI& a ){ *this = *this % a;
       return *this; }
380 inline MpI& MpI::operator>>=( const int n ){ *this = *this >> n;
        return *this; }
381 inline MpI& MpI::operator<<=( const int n ){ *this = *this << n;
        return *this; }
382 inline int operator==( const MpI& a, const MpI& b ){
383     int ll, ls; u_short* p;
384     if( a.l > b.l ){ ll = a.l; ls = b.l; p = a.v;}
385     else{ ll = b.l; ls = a.l; p = b.v; }
386     int i=0;
387     for( ; i<ls; i++ ) if( a.v[i] != b.v[i] ) return 0;
388     for( ; i<ll; i++ ) if( p[i] != 0 ) return 0;
389     return 1;
390 }
391 inline int operator!=( const MpI& a, const MpI&b )
392 { return !( a == b ); }
393 inline int operator>=( const MpI& a, const MpI&b ){
394     int i, ll;
395     if( a.l > b.l ){
396         ll = b.l-1;
397         for( i=a.l-1; i>ll; i-- )
398             if( a.v[i] != 0 ) return 1;}
399     else{ll = a.l-1;
400         for( i=b.l-1; i>ll; i-- ) if( b.v[i] != 0 ) return 0;}
401     for( ; i>=0; i-- )
```

```
402         if( a.v[i] > b.v[i] ) return 1;
403         else if( a.v[i] < b.v[i] ) return 0;
404     return 1;
405 }
406 inline int operator<=( const MpI& a, const MpI& b ){
407     int i, ll;
408     if( a.l > b.l ){
409         ll = b.l-1;
410         for( i=a.l-1; i>ll; i-- ) if( a.v[i] != 0 ) return 0;}
411     else{ ll = a.l-1;
412         for( i=b.l-1; i>ll; i-- )
413             if( b.v[i] != 0 )return 1;}
414     for( ; i>=0; i-- )
415         if( a.v[i] > b.v[i] ) return 0;
416         else if( a.v[i] < b.v[i] ) return 1;
417     return 1;
418 }
419 inline int operator>( const MpI& a, const MpI& b )
420 { return !( a <= b );}
421 inline int operator<( const MpI& a, const MpI& b )
422 { return !( a >= b ); }
423 inline int isEven(const MpI& a ){ return !( a.v[0] & 0x1); }
424 inline MpI exp( const MpI& a, const MpI& b, const MpI& c ){
425     MpI r = 1;
426     if( b == 0 ) return r;
427     MpI aa = a, bb = b;
428     while( bb > 0 ){
429         if( isEven( bb ) ){
430             bb >>= 1; aa *= aa; aa %= c;
431         }else{
432             bb--; r *= aa; r %= c; }
433     }
434     r.adjustLength(); return r;
435 }
436 inline void randInitial(){srand((unsigned)time( NULL));}
437 inline MpI random(const int n ){
438     MpI a( n, 0 );
439     for( int i=0; i<n; i++ ) a.v[i] = (short)rand();
440     return a;
441 }
442 inline MpI random_void(){
443     MpI a;
444     for( int i=0; i<a.l; i++ ) a.v[i] = (short)rand();
445     return a;
446 }
447 inline MpI randomBit( const int l ){
```

```
448     MpI r = random( l / COL_LENGTH + 1 );
449     r >>= r.getBitLength() - l;
450     r.adjustLength(); return r;
451 }
452 inline int isprime( const MpI& n ) {
453     if( isEven(n) ) return 0;
454     MpI n1 = n - 1, s = 0, m = n - 1;
455     while( isEven( m ) ){ m >>= 1; s++; }
456     MpI a = 1;
457     while( a < 2 ){ a = random_void(); while( a > n ) a >>= 1;}
458     MpI y = exp( a, m, n );
459     if( y == 1 ) return 1;
460     if( y == n1 ) return 1;
461     MpI i = 1;
462     while( i < s ){
463         y = exp( y, 2, n );
464         if( y == n1 ) return 1;
465         if( y == 1 ) return 0;
466         i++;
467     }
468     return 0;
469 }
470 inline MpI inv(const MpI& a,const MpI& n ){
471     MpI q, s = a, t = n, u = 1, v = 0, w, x;
472     int qs, ws, ts, ss, vs, us, xs;
473     qs = ws = ts = ss = vs = us = xs = 1;
474     while( ss != 0 && s > 0 ){
475         q = t / s; qs = ts == ss;
476         x = q * s; xs = qs == ss;
477         if( ts ){
478             if( xs )
479                 if( t >= x ){ w = t - x; ws = 1;}
480                 else{ w = x - t; ws = 0; }
481                 else{ w = t + x; ws = 1; }
482         }else{
483             if( xs ){ w = t + x; ws = 0;}
484             else{
485                 if( t >= x ){ w = t - x; ws = 0; }
486                 else{ w = x - t; ws = 1;}
487             }
488         }
489         t = s; ts = ss; s = w; ss = ws;
490         x = q * u; xs = qs == us;
491         if( vs ){
492             if( xs ){
493                 if( v >= x ){ w = v - x; ws = 1;}
```

```
494                 else{ w = x - v; ws = 0; }}
495             else{ w = v + x; ws = 1;}
496         } else{
497             if( xs ){ w = v + x; ws = 0; }
498             else{
499                 if( v >= x ){ w = v - x; ws = 0;}
500                 else{ w = x - v; ws = 1;}
501             }
502         }
503         v = u; vs = us; u = w; us = ws;
504     }
505     if( ! vs ) return n - v;
506     return v;
507 }
508 inline MpI PrimeGen( const int k ){
509     MpI t;
510     do{ t = randomBit( k );
511         if( isEven( t ) ) t--;
512         while( !isprime(t) ) t +=2;
513     }while( t.getBitLength()!= k );
514     t.adjustLength(); return t;
515 }
516 inline void Key_Gen(MpI& p, MpI& g, MpI& s, MpI& h, const int& k
        ){
517     MpI a,b;
518     a = PrimeGen(k-256);a.adjustLength();
519     do{ b = PrimeGen(256);b.adjustLength();p = 2 * a * b + 1;
520     }while(!isprime(p));
521     g = 2; while((exp(g,2,p)==1)||(exp(g,a,p)==1)||(exp(g,b,p)
           ==1)) g++;
522     s = randomBit(100); h = exp(g, s, p);
523     p.adjustLength(); g.adjustLength();
524     s.adjustLength(); h.adjustLength();
525 }
526 inline void Enc(MpI& m, MpI& c_1, MpI& c_2, MpI& g, MpI& h, MpI&
         p){
527     MpI r = randomBit(100); c_1 = exp(g, r, p);
528     MpI c_2_t; c_2_t = exp(h, r, p); c_2 = mul_mod(m, c_2_t, p);
529 }
530 inline MpI mul_mod(const MpI& a, const MpI& b, const MpI& n ){
531     MpI r; r=(a*b) % n; r.adjustLength(); return r;
532 }
533 inline void Mul_C(MpI& c_1, MpI& c_2, const MpI& c1_1, const MpI&
         c1_2, const MpI& c2_1, const MpI& c2_2, const MpI& p){
534     c_1 = mul_mod(c1_1, c2_1, p); c_2 = mul_mod(c1_2, c2_2, p);
535 }
```

```
536 inline MpI Dec(MpI& c_2, MpI& c_1, MpI& s, MpI& p){
537     MpI m_, m_t, m_tt;
538     m_tt = exp(c_1, s, p);
539     m_t = inv(m_tt, p); m_ = mul_mod(c_2, m_t, p);
540     return m_;
541 }
```

付録B　公開鍵p

Table B.1: k ビットの公開鍵 p

k（bit）	生成時間 t_0(s)	原始根 g	p（十六進数）
128	26	2	b49d3751563e487679f239e275a72c3b
256	242	2	f434096481f558aaf18921ff4b5f7d9aeb
512	2952	2	dcaad94b374d4c8a8263821eec063e24 773c4db40421bc3fde598099b62d56488 23c5a3118254c537653b3c5949cd6900f da1383ae4ce3294b2a9c73f9376ae3
1024	8443	5	ca7df0a8288a467e77f88afbb68ffc9c3a1 ee48249fbb48a5cf89f2c22b41b5ac4b46 77757fff4e6a58032a31334bd1a3f0351b 095fc3ceeeb6d9a5b8ee974e4f6fc97a94 d95044de4ff711822254ff415977c1fcd26 cdae4beda5284218ae40b8f422ed22789 edbcae9647bd2d898613d2308515a4a8 e41cf527a4e36043a37
2048	58400	2	a81a02f3da153e9167455930977d7bd83 0251137e5c5c184567d1219dacfe9c057 da3d8aa4bbc7e1a67a15e4818ee1085b 94451e8cb3de9cd50cd49f68cda2aaa37 8c72e673de14c9fb054eba487e567bf09 af765f6379e4983dd01416e338994ba5f2 1225676a83f2f2c7551d49606e6fef9674 a2c20c951cc9e3fb7fb04dd132ef15dfc32 ebdd1e5b6a614ce64cf2fdab144925c80 10a7ac7d30f5d02df4f1b4db902e8240b9 7c4fcbe0498c386eab1abc63e39dc0e33 668ad76b7aee751e79306a3bf56a68c1b 3eadeab2c6e8472f5384f59dcd9c70be7 4b728f8040f5862e188c8c4783456eb38 6e98a70586e4ad312ed3a95b35c86aa6 49d9ab42d07e77

付録C　制御入力$\bar{u}(t)$

Table C.1: $\gamma_y = 1 \times 10^4$ の場合の $\bar{u}(t)$

t(s)	$u(t)$	$\bar{u}(t)$	t(s)	$u(t)$	$\bar{u}(t)$
0	-5.3	-5.3	2.6	0.0988662	0.09858
0.1	-4.9526751	-4.95232	2.7	0.1026981	0.10229
0.2	-4.0753502	-4.07517	2.8	0.0927288	0.09222
0.3	-2.9128694	-2.91235	2.9	0.0737177	0.07367
0.4	-1.6832111	-1.68275	3.0	0.05035	0.05035
0.5	-0.5571148	-0.55703	3.1	0.026659	0.0265
0.6	0.3511886	0.35086	3.2	0.0056816	0.0053
0.7	0.9838231	0.98368	3.3	-0.0106477	-0.0106
0.8	1.3321232	1.33189	3.4	-0.0214809	-0.0212
0.9	1.4251382	1.42464	3.5	-0.026871	-0.0265
1.0	1.3163981	1.31599	3.6	-0.0275229	-0.02703
1.1	1.0709816	1.0706	3.7	-0.0245708	-0.02438
1.2	0.7544179	0.75419	3.8	-0.0193079	-0.01908
1.3	0.4244081	0.424	3.9	-0.0129691	-0.01272
1.4	0.1256047	0.12508	4.0	-0.0066303	-0.00636
1.5	-0.1126303	-0.11236	4.1	-0.0010812	-0.00106
1.6	-0.2759975	-0.2756	4.2	0.00318	0.00265
1.7	-0.3632037	-0.36305	4.3	0.0059625	0.00583
1.8	-0.3827819	-0.38266	4.4	0.0072875	0.00689
1.9	-0.3495509	-0.34927	4.5	0.007367	0.00689
2.0	-0.2811385	-0.2809	4.6	0.0065031	0.00636
2.1	-0.1950665	-0.19504	4.7	0.0050509	0.00477
2.2	-0.1065989	-0.10653	4.8	0.0033337	0.00318
2.3	-0.0273851	-0.02703	4.9	0.0016377	0.00159
2.4	0.0350224	0.03498	5.0	0.0001749	0
2.5	0.0771362	0.07685			

Table C.2: $\gamma_y = 1 \times 10^5$ の場合の $\bar{u}(t)$

t(s)	$u(t)$	$\bar{u}(t)$	t(s)	$u(t)$	$\bar{u}(t)$
0	-5.3	-5.3	2.6	0.0988662	0.098845
0.1	-4.9526751	-4.952638	2.7	0.1026981	0.102661
0.2	-4.0753502	-4.075329	2.8	0.0927288	0.092697
0.3	-2.9128694	-2.912827	2.9	0.0737177	0.07367
0.4	-1.6832111	-1.683174	3.0	0.05035	0.05035
0.5	-0.5571148	-0.557083	3.1	0.026659	0.026659
0.6	0.3511886	0.351178	3.2	0.0056816	0.005671
0.7	0.9838231	0.983786	3.3	-0.0106477	-0.0106
0.8	1.3321232	1.332102	3.4	-0.0214809	-0.021465
0.9	1.4251382	1.425117	3.5	-0.026871	-0.026871
1.0	1.3163981	1.316361	3.6	-0.0275229	-0.027507
1.1	1.0709816	1.070971	3.7	-0.0245708	-0.024539
1.2	0.7544179	0.754402	3.8	-0.0193079	-0.019292
1.3	0.4244081	0.424371	3.9	-0.0129691	-0.012932
1.4	0.1256047	0.125557	4.0	-0.0066303	-0.006625
1.5	-0.1126303	-0.112625	4.1	-0.0010812	-0.00106
1.6	-0.2759975	-0.275971	4.2	0.00318	0.003127
1.7	-0.3632037	-0.363156	4.3	0.0059625	0.005936
1.8	-0.3827819	-0.382766	4.4	0.0072875	0.007261
1.9	-0.3495509	-0.349535	4.5	0.007367	0.007367
2.0	-0.2811385	-0.281112	4.6	0.0065031	0.006466
2.1	-0.1950665	-0.19504	4.7	0.0050509	0.005035
2.2	-0.1065989	-0.106583	4.8	0.0033337	0.003286
2.3	-0.0273851	-0.027348	4.9	0.0016377	0.00159
2.4	0.0350224	0.03498	5.0	0.0001749	0.0001749
2.5	0.0771362	0.077115			

Table C.3: $\gamma_y = 1 \times 10^7$ の場合の $\bar{u}(t)$

t(s)	$u(t)$	$\bar{u}(t)$	t(s)	$u(t)$	$\bar{u}(t)$
0	-5.3	-5.3	2.6	0.0988662	0.0988662
0.1	-4.9526751	-4.9526751	2.7	0.1026981	0.1026981
0.2	-4.0753502	-4.0753502	2.8	0.0927288	0.0927288
0.3	-2.9128694	-2.9128694	2.9	0.0737177	0.0737177
0.4	-1.6832111	-1.6832111	3.0	0.05035	0.05035
0.5	-0.5571148	-0.5571148	3.1	0.026659	0.026659
0.6	0.3511886	0.3511886	3.2	0.0056816	0.0056816
0.7	0.9838231	0.9838231	3.3	-0.0106477	-0.0106477
0.8	1.3321232	1.3321232	3.4	-0.0214809	-0.0214809
0.9	1.4251382	1.4251382	3.5	-0.026871	-0.026871
1.0	1.3163981	1.3163981	3.6	-0.0275229	-0.0275229
1.1	1.0709816	1.0709816	3.7	-0.0245708	-0.0245708
1.2	0.7544179	0.7544179	3.8	-0.0193079	-0.0193079
1.3	0.4244081	0.4244081	3.9	-0.0129691	-0.0129691
1.4	0.1256047	0.1256047	4.0	-0.0066303	-0.0066303
1.5	-0.1126303	-0.1126303	4.1	-0.0010812	-0.0010812
1.6	-0.2759975	-0.2759975	4.2	0.00318	0.00318
1.7	-0.3632037	-0.3632037	4.3	0.0059625	0.0059625
1.8	-0.3827819	-0.3827819	4.4	0.0072875	0.0072875
1.9	-0.3495509	-0.3495509	4.5	0.007367	0.007367
2.0	-0.2811385	-0.2811385	4.6	0.0065031	0.0065031
2.1	-0.1950665	-0.1950665	4.7	0.0050509	0.0050509
2.2	-0.1065989	-0.1065989	4.8	0.0033337	0.0033337
2.3	-0.0273851	-0.0273851	4.9	0.0016377	0.0016377
2.4	0.0350224	0.0350224	5.0	0.0001749	0.0001749
2.5	0.0771362	0.0771362			

謝辞

本研究を進めるにあたり，ご指導，ご鞭撻を賜りました指導教員の小木曽公尚准教授と新誠一教授に心より感謝いたします．また，研究室の先輩，同期と後輩，それに生活を支えてくださった家族，友人とバイト先の店長に心から感謝します.